Patterns for E-commerce applications

Gustavo Rossi *, Fernando Lyardet *, Daniel Schwabe **
*LIFIA Facultad de Informática. UNLP.
La Plata, Argentina
E-mail: {fer,gustavo}@sol.info.unlp.edu.ar
**Departamento de Informática, PUC-Rio, Brazil
E-mail: schwabe@inf.puc-rio.br

Abstract

In this paper we present some patterns we found in E-commerce applications. First, we briefly characterize these applications as a particular case of Web applications. We next review some Web patterns that can be used in E-commerce applications. Finally, we present five new patterns: Opportunistic Linking, Advising, Explicit Process, Easy Undo and Push Communication.

Introduction

The World Wide Web has become a popular platform for E-commerce applications. These applications combine navigation through an electronic catalogue with operations affecting this catalogue. In this sense e-commerce applications are a particular kind of Web applications with similar requirements: good navigational structures, usable interfaces, a clear domain model, etc.

However, E-commerce applications present new challenges to the designer: we not only need to help the user find what he wants (a product he will buy) but also ease the shopping process. For example we should keep him informed about new releases and, last but not least, keep him in the electronic shop for a longer time.

We have developed a conceptual framework for reasoning on design reuse in Web applications. In our approach [Schwabe98, Schwabe99], applications have an object model, a navigational view and the interface. While the object model deals with the usual application behavior, the navigational view defines the architecture of the hyperspace and the interface defines the look and feel of the application. While it is obvious that e-commerce applications will involve particular problems at the object level (related for example with the particular workflow of activities), we are mainly interested in design structures at the navigational level. In this paper we discuss some particular problems related with the navigation topology of a virtual store.

We have been mining patterns for designing Web applications for the last 5 years [Rossi96, Rossi97,Lyardet98, Lyardet99, Rossi99]. These patterns provide guidelines to organize the information hyperspace and to design usable interfaces. In some way, they are similar to Alexandrian patterns, as their aim is to assist the designer in the process of building confortable information spaces, where users can navigate or search information. Navigation patterns show how to push forward the basic hypermedia paradigm (on top of which we build Web applications), based on nodes, links and indexes and build more complex navigation arquitectures.

Some of the papers we mined before (See [Rossi99]) can be easily applied to the E-commerce field. For example the ***News*** pattern shows how to make the user aware of new releases by dedicating part of the home page to those news. The ***Basket*** pattern provides a way to postpone operations on products by putting them in a (shopping) basket. Finally the ***Landmark*** pattern helps to organize the site in sub-sites (shops) that are reachable from every node.

In this paper, we present other patterns that can be used to build successful Web applications. It is interesting to notice that while applying some of them, we solve an usual customer problem (how to find what he wants), others help to solve a problem of the store (how to get the user “seduced” with the store and not leave it). We will point those differences in each pattern.

Opportunistic Linking

Intent
Keep the user interested in the site. Seduce him to navigate in the site even when he has already found what he was looking for

Motivation
Suppose we are building a virtual store such as www.amazon.com. By entering the site we can buy many different products such as videos, books or CDs. We can explore the products, and besides we provide links to recommendations, comments about the products, news, etc. However, many users navigate with a specific target: for example buying one particular book. Once they have bought that book, they may leave the site.

One possibility is to add links to each product page to motivate the reader to navigate to other products. In a well-structured site, however, we must try to provide links with strong semantics to reduce the risk of disorientation. So, how do we reconcile these two requirements?

Forces
- We want to keep the user navigating in our store even after he bought something
- We don't want to compromise the structure of the store by adding not meaningful links

Solution
Improve the linking topology by suggesting new products to explore from a given one. Use relationships with strong semantics to make the user feel confortable. Take into account that many of these links may change from day to day so that the interface should be defined accordingly. Notice that this pattern can also be used at the conceptual level to derive new relationships. However the intent is clearly navigational: keep the user navigating in a pleasant way.

Examples
Opportunistic Linking can be found in many electronic stores. For example in www.cdnow.com or www.amazon.com you may find CDs related with the one you chose. In Figure 1 we show an extreme example in Amazon.com. Once the user has chosen a book and put it in the shopping card, he receives a suggestion of another book he might be interested in.

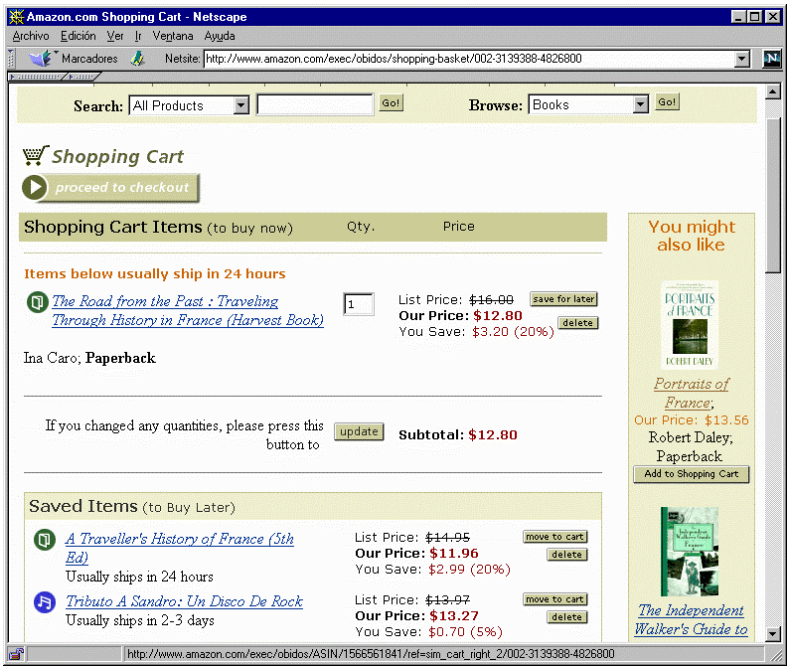

Figure 1: Opportunistic linking in amazon.com

Consequences

- Keep the user engaged in the site; as an indirect consequence improves sells
- Too much linking (may cause cognitive overhead)
- More complex underlying data model

Implementation

The implementation of this pattern is not complex as it does not add anything new to a conventional Web application. We just need to adapt the interface by showing more links (as in Figure 1). Those links can be calculated directly from the underlying data model using algorithms for finding "similarities". These algorithms can be straightforward: for example finding some books with simple key-word matching (the same subject, e.g. travel) or more complex as discussed in the pattern Advising.

Related Patterns

Opportunistic Linking is quite similar to Advising; in fact one may argue that what you are doing when applying this pattern (see Figure 1) is advising the customer. However, the intent is different. While opportunistic linking tries to keep the user inside the store by giving him new ideas to buy, Advising helps him choose what he wants.

Advising

Intent
Help the user find a product in the store. Assist him according to his wishes

Motivation
Many times users enter into a virtual store just to find some product they would like to buy. In a typical store there may be thousands of products and providing good indexes or search engines may be just a partial solution to give him some orientation. Applying the News pattern [Rossi99] we can show him new products or releases. However, news may change from day to day and besides we can not be sure that he will be interested in a new product.

Forces
- Customers in an electronic shop may need to be assisted to find a product
- Search engines and indexes (for example taxonomic) are useful but they may be an incomplete solution
- We should take into account what the user may want

Solution
Build specific functionality for advising about products. This functionality may be implemented in different ways. For example, there can be a complete subsystem for recommendations as in www.amazon.com that use customer's profiles (in general their buying history) for recommending products. It may be more general, and present the user the best seller products, or products on sale, etc. The design of the advising facilities should not interfere with the global navigational structure

Examples
Advising is used in almost all virtual stores. For example, Amazon not only provides recommendations according to the user's profile but includes best sellers, updating hourly its list of 100 hot books. In www.barnesandnoble.com for example a bargain section is included in the home page together with general recommendations. www.netgrocer.com just gives information about products on sale. In Figure 2 we show the structure of advising in www.cdnow.com. In www4.activebuyersguide.com one can find a complete site devoted to advising. Many E-commerce sites provide links to this site to provide "unbiased" descriptions of their products.

Figure 2: Advising in www.cdnow.com

Consequences

- Help customers decide what to buy
- May help to induce customers towards certain products
- It is necessary to keep the user's profile and history
- Requires complex algorithms

Implementation

There are different ways to implement Advising. The most simple one is to include simple recommendations in a home page (like one can find in, for example, Amazon.com). This implementation only needs those recommendations (for example top sellers) to be shown in the page. A more subtle implementation keeps track of the user's profile (in the underlying database) by recording what he bought in the past and suggesting similar products. The interface is straightforward as it is similar to a "conventional" page.

Related patterns

Advising is similar to Opportunistic Linking. However, in this example the intent is to help the user find its way towards a product, while Opportunistic Linking is aimed at "trapping" the user once he found a product, Advising has a more general scope. In fact, both patterns could be considered as specific versions of a more generic one. Advising can also be combined with the News patterns [Rossi 99].

Explicit Process

Intent
Help the user understand the buying process when it is not atomic

Motivation
In most virtual stores the checkout process is complex and involves filling different forms (shipping and billing address, information on credit card, etc). Taking into account the very nature of the Web the user may experience disorientation and may wonder if he had already filled some information and may be confused about the whole process. This may cause that he either cancel the process or he tries to backtrack to previous pages, perhaps leading to some inconsistent state (See Easy Undo).

Forces
- The checkout or registering process may be complex
- Users tend to feel disoriented as they progress through this process

Solution
Give the user a perceivable feedback about the process by keeping him up to date about which steps he has already accomplished. This can be done by using a “progress” line or by just enumerating the steps and informing where he is now. Take into account the possible consequences of his backtracking (See Easy Undo) in order to minimize the possibilities of reaching inconsistent states.

Examples
In most stores we can find simple implementations of this pattern. For example in **www.barnesandnoble.com** for example, the customer has to progress through 7 steps that are clearly indicated in a sequential way. In **www.amazon.com** meanwhile the implementation is more elegant as shown in Figure 3. In the top of the screen you can see a process line indicating that you are in step “Items” and that you still lack four steps until finishing.

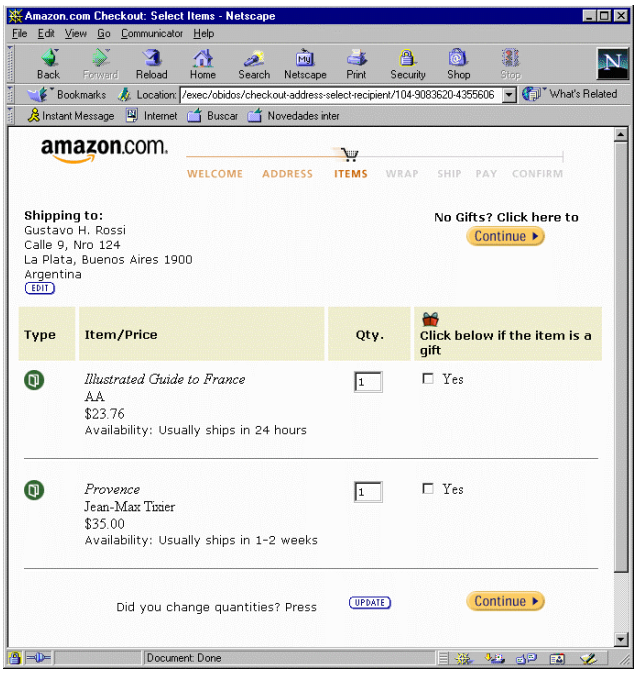

Figure 3: Explicit Process in Amazon.com

Consequences

- Helps to understand complex processes
- May help to simplify the undoing process
- It is necessary to keep user's state (in a session)
- May complicate the user interface

Implementation

The implementation of this pattern is fairly simple, since it does not deal with the business logic but rather provide a visual feedback to the user of the current process. Even though it is possible to perform some client side solution, the information to be shown is usually provided by the sever and it is implemented either by plain html or a combination of html and graphics

Related Patterns:

Explicit Process is related with Easy Undo as they both tend to simplify and make the checkout (or other non-atomic) process safer. Explicit Process is a particular case of Process FeedBack [Rossi00]

Easy Undo

Intent
Provide safe undoing capabilities in a complex process

Motivation
As previously explained some process in e-commerce applications may be quite complex (as the checkout process). While indicating the step in which we are is useful, it may also happen the user misspelled some information or he may want to change some data in his order. The naive solution would be letting the customer backtrack (using his browser Back button), correct the corresponding data and redo the process. However, as he may not be aware of the exact browsing semantics, it may happen that he finds an inconsistent state. For example, the shopping basket may have changed (if some items were changed for example). The problem with the Back button is that the semantic of backtrack can interfere significantly with the intended undo operation.

Forces
- The checkout or registering process may be complex
- The customer may need to undo some previous operation
- Using the Back button may yield unexpected results

Solution
Provide the user with Undo facilities avoiding him to use navigation facilities for this purpose. The undo facilities will have to take into account the customer state in the process in order to be effective. Take into account the browsing semantics in order not to reach an inconsistent state. This pattern extends the idea of backtracking typical in Web applications adapting it to the application's semantics. Instead of returning to the last Web page (using the Back button), we return to the corresponding state to undo the operation. It is important to stress that this difference is specific to this domain as the Web is based on a simple hypertext paradigm with a general backtrack functionality.

Examples
We have found many different examples and implementations of this pattern in E-commerce applications. For example in **www.powells.com**, a customer is exposed to all the information in the confirmation step and he can choose to update or modify your data. Meanwhile in **www.amazon.com** (See Figure 4) customer information is added incrementally and he can choose to change previously entered data. Notice the small buttons near each information item, indicating that you can correct the information. When you select to edit the billing address for example, you return to the corresponding page, and once changed you can continue the process. Notice that we are using a backtrack-forward algorithm that changes the usual browsing semantics adapting it to the needs of the store.

Consequences
- Simplifies undoing
- Helps to solve the tension between backtracking and undoing
- Requires an elaborated interface
- May need to keep the user's state

Implementation
As with many other implementation issues in Web applications, the strategy adopted to solve a problem has many variants due to the inherent conversational nature of interaction in e-commerce. The record of a particular customer activity (the "memory" of a vendor-client conversation, or "state" of such interaction) is a key component to implement the solution. Such "state" can be either Centralized, OnTheWire or Client-Side according to the developers decision on the particular location of the information. Once the information about the user activity is recovered (-regardless the strategy being used-), the actual implementation is

reduced to the simple problem of establishing the right transition to the particular instant. Finally, there are several possibilities for the implementation of the necessary logic. It may reside as a script of a DHTML page sent to the client. Another possibility is through a server side code (as an ASP, JSSP, JSP, Perl and other scripting language program, or as a CGI, ISAPI http server extension or finally as a business rule wrapped in a component –either COM or EJB-). Finally, other server based solutions are based on server application frameworks like Borland Delphi's Midas/Webbroker which support http and other internet protocols directly. See the picture below.

State / Logic	Server-Based		Client-Based		OnTheWire	
Server-side	JSSP JSP / EJB ASP / COM CGI/ISAPI	EJB ASP-COM JSS Web Frameworks Database	JSSP JSP / EJB ASP / COM CGI/ISAPI	Cookies XML	JSSP JSP / EJB ASP / COM CGI/ISAPI	HiddenFields Fat URLs
Client-Side	Vbscript Javascript Java Applet	RDS Javascript	Vbscript Javascript Java Applet	Cookies XML	Vbscript Javascript Java Applet	HiddenFields Fat URLs

Related Patterns

Easy Undo can be used jointly with Explicit Process keeping the user aware of his current state in the process. It is also a special case of Easy Undo.

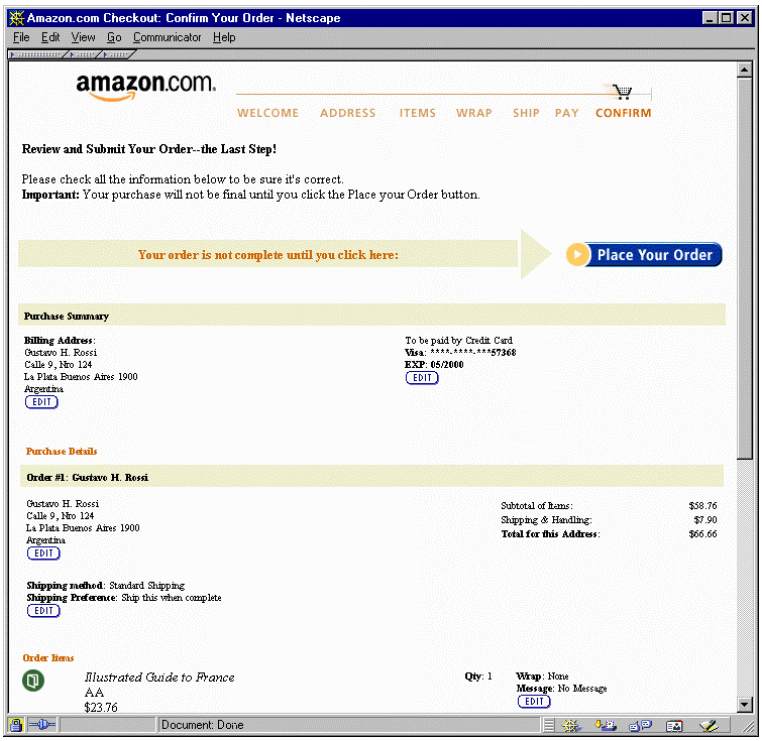

Figure 4: Easy Undo in Amazon

Push Communication

Intent
Simplify the searching process for customer-selected areas or products

Motivation
It is well known that the Web implements a so-called “pull” model. Customers visit a virtual store and pull the information by browsing through the information base. While this model works fine in most cases, there are situations in which it puts excessive burden on the customer. Suppose that he is interested in books on a particular subject or is organizing a trip to a certain place. How can we simplify the process of his finding that information? The naive, pull-based, solution is letting the user enter each day, find the corresponding information to see if it is what he wanted. Even when using the News pattern [Rossi99] it may happen that not all new products are announced. We could also use the Advising pattern and add each new product that may interest the customer to the list of advised ones. However, even in this case, the customer has to enter the store and find if the information is there.

Forces
- Finding new information is not always easy
- User waste a lot of time connecting to find new products
- The basic Web model is pull-based

Solution
Combine the usual Web pull model with a push-based communication. Find ways to communicate with the customer without forcing him to find the information. This solution may have different implementations; for example the customer may subscribe to a given subject (or type of product) and receive a mail message each time a new item matching his interests has arrived. This email may directly contain the URL of the product for him to “pull” the information. Another possible alternative is to personalize his site using “channels”. Once he enters into a channel he finds the information on the content he is willing. Mailing and channels can be combined to improve customer access to the information. Notice that this pattern goes further in the usual communication metaphor between Web users and Web sites.

Examples
Most virtual stores provide some kind of subscription mechanism for helping customers know when a particular item has entered into the shop. In **www.amazon.com** for example, each time a customer search for a product (in a particular subject or area) he is invited to sign up to receive mails each time a new release arrives (See Figure 5). In **www.izero.com**, and **www.netzero.com** that provide free Internet access, Push Communication is used to send advertising to the user. Finally in Figure 6 we show the Expedia.com fare tracker. The customer selects an itinerary and he can receive feedback on cheap fares both by email or by having his profile updated (a kind of channel). In this example the customer may choose to go to his personal profile to have information related to his preferences. Finally in **www.bloomeberg.com** we can see a more extreme example of Push Communication. You can select some stocks and you receive (in a separate window) information about changes in those stocks in real time.

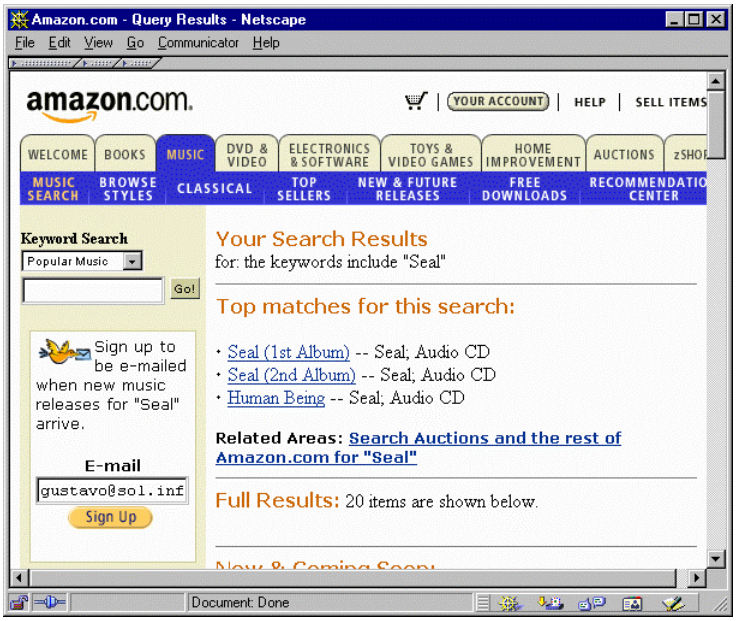

Figure 5: Subscribing to a push service in Amazon

Figure 6: Push Communication in Expedia.com

Consequences

- Simplifies user's tasks by providing precise information about his needs
- Is useful for advertising new products
- It is necessary to keep the user's profile
- Must rely on a subscription/update mechanism

Implementation

Push communication is usually implemented using a client-side strategy. Although this may sound awkward, it help address several issues related with the instability of communication links and users behaviors. Client-side scripts embedded in DHTML pages sent to the customers regularly retrieve the information form the server and refresh customer contents providing the user perception of a push communication. Also, this strategy is the used when Java applets are used on client pages, as in the example available in http://www.bloomberg.com/help/mon_jump.html?topnav=front.

On the other hand, server-based implementation is also possible thanks to the http 1.1 property of "keep alive" connections, that allows the server redirect current alive connections to other pages with updated information and thus refreshing clients contents

Related Patterns

Push Communication may be considered as a particular implementation of the News pattern [Rossi99]. However, we preferred to differentiate these two patterns as they comprise quite different strategies for managing customer-shop communication and they involve different technologies.

The discussion underlying Push Communication can be rooted to the Observer design pattern. In fact we can consider the customer as an observer on a set of facets of the store and the communication (for example an email) as a particular implementation of the usual "update" message, typically found in object-oriented implementations. Once the observer received the email he pull the information. The discussion on granularity of changes can also be rooted to the discussion in [Gamma95].

Concluding Remarks

We have presented five new patterns for electronic commerce applications. These patterns focus mainly on ways to solve usual problem customers have to find and buy products in the shop. They provide hints to the Web application designer in order to make these applications more usable and effective both from the point of customers and owners of the store. By showing non-trivial extension of the basic Web model (based on nodes and links) these patterns help to improve the navigation topology and some aspects of the customer-store communication 'styles.

Bibliography

[Gamma95] Gamma E.,Helm R., Johnson R., Vlissides J."Design Patterns. Elements of Reusable Object Oriented Software.", Addison Wesley, 1995.

[Lyardet99] F. Lyardet, G. Rossi, D. Schwabe: "Patterns for adding search capabilities to Web Information Systems". Proceedings of EuroPLoP 99.

[Rossi96] G. Rossi, A. Garrido, S. Carvalho: "Patterns for object-oriented hypermedia applications". In Pattern Languages of Programs II, Addison Wesley, 1996.

[Rossi 97] G. Rossi, D. Shwabe and A. Garrido : Design Reuse in Hypermedia Design Applications Development Proceedings of ACM International Conference on Hypertext (Hypertext'97), Southampton, UK, 1997, ACM Press.

[Rossi99] G. Rossi, D. Schwabe and F. Lyardet: "Patterns for designing navigable information spaces". Pattern Languages of Programs IV, Addisson Wesley, 1999.

[Rossi00] G. Rossi, D. Schwabe and F. Lyardet: "User Interface Patterns for Hypermedia Applications". Proceedings of AVI00, Advanced Visual Interfaces, Palermo, Italy, May 2000.

[Schwabe98] D. Schwabe, G. Rossi: "An object-oriented approach for Web-based applications design". Theory and Practice of Object Systems (TAPOS), October 1998.

[Schwabe99] D. Schwabe, G. Rossi, F. Lyardet: "Web application models are more than conceptual models". Proceedings of the First International Workshop on the WWW and Conceptual Modeling, Paris, November, 1999.

Appendix: Glossary of patterns and relationships among patterns

News:
Given a large and dynamic Web application provide the users with information about new items that have been added

Basket:
Keep track of user selections during navigation, making these selections persistent to process them when the user decides it

Landmark:
Provide easy access to different though unrelated subsystems in a Web application.

Process Feed-back:
Keep the user informed about the status of the interaction in such a way that he knows what to expect

Observer:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically

Easy Undo

Relationships among Patterns